CompSci 101 Sec01 **Exam 3** **Fall 2025**

NAME (print): _____

Netid: _____

Which Lecture section are you in:(01 at 10:05am, or 02 at 3:05pm?): _____

Community Standard Acknowledgement (signature): _____

Do NOT spend too much time on any one question.

YOU MUST WRITE YOUR NETID on EVERY PAGE THAT IS LOOSE.

Before starting, make sure your test contains 14 numbered pages, followed by the Python reference sheets. ALL PAGES must be turned in, including the reference pages.

In writing code you do not need to worry about specifying the proper `import` statements. Assume that all libraries we've discussed are imported in any code you write.

If you need more space, there are three blank pages you can write on, but you must indicate you have done so.

DO NOT WRITE any work to be graded on the Python Reference sheets.

Do **NOT** discuss this test **with anyone** until the test is handed back.

Problem	Points	Estimated Minutes
Problem 1	16 pts.	6
Problem 2 (3 parts)	18 pts.	14 (4-6 mins each)
Problem 3 (5 parts)	45 pts.	45 (8-10 mins each)
TOTAL:	79 pts.	65 min

NETID: _____

PROBLEM 1 : (***What is the output? (16 pts) (8 minutes)***)

For the following code, write the output to the right of each print statement.

```
                                                    OUTPUT:
lista = ['bcd', 'axy', 'bop', 'abc']
print(sorted(lista))
#-------------------------------------------------

lista = ['head', 'shoulders', 'knees', 'toes']
listb = [(max(w), w) for w in set(lista)]
print(sorted(listb))
#-------------------------------------------------

lista = ['walrus', 'elephant', 'hippo', 'narwhal']
listb = sorted(lista, reverse=True)
print(listb)
#-------------------------------------------------

lista = ['couch', 'chair', 'seat', 'arm', 'sofa']
listb = sorted(lista, key=len)
print(listb)
#-------------------------------------------------

lista = [(9,1,5),(2,7),(8,2,5)]
listb = sorted(lista, key=max)
print(listb)
#-------------------------------------------------

lista = [(6,3,5),(8,1,4),(6,0,5)]
listb = sorted(lista, key=lambda x:x[1])
print(listb)
#-------------------------------------------------

lista = [(6,0,5),(8,1,4),(6,0,2)]
listb = sorted(lista, key=lambda x: (x[2],x[0]))
print(listb)
#-------------------------------------------------

d = {4: ['S', 'A', 'W'], 5: ['H'], 3: ['C', 'D']}
ans = sorted(d.items(), key=lambda x:x[0]+len(x[1]))
print(ans[0])
```

PROBLEM 2 : (*Short problems (18 pts) (14 minutes)*)

This problem has three parts. Your functions should work for any valid data, not only the examples shown.

PART A (6 pts) (4 minutes)

Consider the following function named `mysteryA` that has two parameters, a dictionary named `adict` in which each key is an integer and each value is a string, and a list of strings named `alist`.

```
1 def mysteryA(adict, alist):
2     ret = set()
3     for val in adict.values():
4         if val in alist:
5             for key in adict:
6                 if adict[key] == val:
7                     ret.add(key)
8     return ret
```

QUESTION 1

Consider that `adict` and `alist` are as shown below:

```
adict = {7:  'gorilla', 3:  'cat', 14:  'shoe', 7:  'cat'}
alist = 'the cat and gorilla wore sandals'.split()
```

What does `mysteryA(adict, alist)` return? (Order does not matter.)

QUESTION 2

Explain in words what the function `mysteryA` does? (Briefly, in 1-2 sentences)

QUESTION 3

For the code above, show below how `adict.items()` can be used to simplify the function; in particular, remove the nested for-loop at line 5. Rewrite the function using `adict.items()` below. The function should always return the same as the code above.

```
def mysteryA(adict, alist):
    ret = set()




    return ret
```

PART B (6 pts) (4 minutes) Consider the following function named `mysteryB` that has two parameters, a string named `word` representing a word, and a string named `sentence` containing words separated by single spaces.

```
1 def mysteryB(word, sentence):
2     ret = set()
3     for someword in sentence.split():
4         diff = set(word) & set(someword)
5         ret.add( len(diff) )
6     return sorted(ret)
```

QUESTION 1

Consider that `word` and `sentence` are defined below.

```
word = "soundsystem"
sentence = 'you switch the engine on'
```

For this input, what does `mysteryB(word, sentence)` return?

QUESTION 2

Explain in words what the `mysteryB` function does. (Briefly, in 1-2 sentences)

QUESTION 3

Suppose we want to sort the list of integers so that integers with the same number of digits are ordered in decreasing order. Which line of code you would modify and what would you change it to? For example, `[9, 6, 3, 89, 43, 21, 521, 101]` is in this order.

NETID: _____

PART C (6 pts) (6 minutes)

Write the function named `enrollments` that has one parameter named `lista` that is a list of tuples, where each tuple has three items: a string (department abbreviation), an integer (course number), and an integer (enrollment).

This function returns a list of the tuples sorted in the following way:

1. Sort in decreasing order of enrollments;
2. If two courses have the same enrollments, then order them by department in alphabetical order;
3. If two courses have the same department and enrollment, then break the tie by placing the smaller course number before the other.

For example, the call `enrollments(lista)` with

```
lista = [('CS', 101, 104), ('MATH', 232, 45), ('CS', 94, 42),
('MATH', 212, 45), ('CS', 330, 154), ('PHIL', 374, 56)]
```

returns the list:

```
('CS', 330, 154), ('CS', 101, 104),  ('CS', 94, 56),
('PHIL', 374, 56), ('MATH', 212, 45), ('MATH', 232, 45)]
```

Complete the function below.

```
def enrollments(lista):
```

NETID: _____

BLANK PAGE IF YOU NEED MORE SPACE (MUST TURN IN)
THERE ARE STILL MORE PROBLEMS TO SOLVE ....

NETID: _____

PROBLEM 3 : (*Wednesday Schedules (48 pts) (48 minutes)*)

This problem is about data related to students' Wednesday schedules in fall 2025.

There are five functions to write in this part. Your functions should work for any valid data, not just the examples shown.

Most of the problems have `datalist` as one of the parameters. The parameter datalist is a list of lists, with each inner list representing information about one student the courses they take on Wednesdays. More specifically, each inner list has:

1. a string representing a student name,
2. an integer representing the student's id number,
3. a string representing the home department of the student's major, which is possibly "Undeclared",
4. a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

For example, assume `datalist` is the lists of lists shown below. The first inner list represents the student named Shah, whose student id number is 342128, has not yet declared their major, and has three courses on Wednesdays: first they have a CompSci course, then an Econ course, and lastly another CompSci course.

Assume that each student name is unique and appears only once in `datalist`, and that each student is undeclared or has one major. Students have between zero and eight courses on Wednesdays.

```
datalist = [
    ['Shah', 342128, 'Undeclared', ['CompSci', 'Econ', 'CompSci']],
    ['Singh', 465621, 'CompSci', ['Econ', 'Latin', 'CompSci', 'Dance']],
    ['Gates', 126884, 'Econ', ['CompSci', 'Math', 'Math', 'Econ']],
    ['Bowman', 634645, 'Psy', ['Econ', 'Psy', 'CompSci', 'CompSci']],
    ['French', 227712, 'Psy', ['Psy', 'Math', 'Psy', 'Math']],
    ['Dole', 937712, 'Dance', ['Math', 'Dance', 'Spanish', 'CompSci']],
    ['Cook', 743953, 'CompSci', [ ]],
    ['Earle', 335672, 'CompSci', ['CompSci', 'CompSci', 'CompSci']],
    ['Nixon', 521672, 'Undeclared', ['Latin', 'Econ', 'CompSci', 'Econ', 'Psy']]
]
```

In solving a part of this problem, **you may call any of the functions of preceding parts**. For example, Part A must be solved on its own, Part C can call the functions from Parts A and B, and so on.

Go to the next page to start Part A of this problem.

NETID: _____

Part A (9 pts) (9 minutes)

Write the function named **`beginsOrEndsWith`** that has two parameters named **`datalist`**, which is a list of lists in the format described earlier, and **`dept`**, a string that is a department at Duke.

We repeat the format of parameter `datalist`, which is a list of lists. Each inner list has: (1) a string representing a student name, (2) an integer representing the student's id number (3) a string representing the home department of the student's major (possibly "Undeclared"), and (4) a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

This function returns the number of students whose Wednesdays begin or end (possibly both) with a course from the parameter department **`dept`**.

For example, using the `datalist` on the first page of this problem, the call `beginsOrEndsWith(datalist, 'CompSci')` returns `5` since there are five students whose first and/or last courses are CompSci courses.

Complete the function below.

```
def beginsOrEndsWith(datalist, dept):
```

NETID: _____

Part B (9 pts) (9 minutes)

Write the function named **`countMajorMatches`** that has one parameter named **`datalist`**, which is a list of lists in the format described earlier.

We repeat the format of parameter `datalist`, which is a list of lists. Each inner list has: (1) a string representing a student name, (2) an integer representing the student's id number (3) a string representing the home department of the student's major (possibly "Undeclared"), and (4) a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

This function returns a **sorted list** of tuples, where each tuple has two items. The first item is a string that is the name of a student **with a declared major**, and the second item is an integer that is the number of courses on Wednesdays in the same department as their major. **No student with an undeclared major is represented in the returned list.** The tuples are sorted first by the second item, the count, in increasing order. Ties are broken by the students' orders in `datalist`.

For example, using the `datalist` described on the first page of this problem, the call `countMajorMatches(datalist)` returns the list of tuples:

`[('Cook', 0), ('Singh', 1), ('Gates', 1), ('Bowman', 1), ('Dole', 1), ('French', 2), ('Earle', 3)]`.

```
def countMajorMatches(datalist):
```

NETID: _____

Part C (9 pts) (9 minutes)

Write the function named **`takenMostAt`** that has two parameters, one named **`datalist`**, which is a list of lists in the format described earlier, and one named **`num`**, which is a value between 0 and 7.

We repeat the format of parameter `datalist`, which is a list of lists. Each inner list has: (1) a string representing a student name, (2) an integer representing the student's id number (3) a string representing the home department of the student's major (possibly "Undeclared"), and (4) a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

This function returns a **sorted list** of **distinct** departments with the most students that take a course from their department as the **`num`**'th course in their Wednesday schedule (if any); if there is more than one department in the list, they are sorted in alphabetical order.

For example, using the `datalist` described on the first page of this problem, the call `takenMostAt(datalist, 1)` returns `['Econ', 'Math']` since, for each of Econ and Math, there are two students that take their courses at index 1 in their schedules and all other departments have 0 or 1 students taking a course in those departments at that index. Similarly, `takenMostAt(datalist, 3)` returns `['CompSci', 'Econ']`.

```
def takenMostAt(datalist, num):
```

NETID: _____

BLANK PAGE IF YOU NEED MORE SPACE (MUST TURN IN)
THERE ARE STILL MORE PROBLEMS TO SOLVE ....

NETID: _____

Part D (9 pts) (9 minutes)

Write the function named **`enrollmentsByKind`** that has one named **`datalist`**, which is a list of lists in the format described earlier.

We repeat the format of parameter `datalist`, which is a list of lists. Each inner list has: (1) a string representing a student name, (2) an integer representing the student's id number (3) a string representing the home department of the student's major (possibly "Undeclared"), and (4) a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

This function returns a dictionary mapping each department to a list of two integers. The first integer (at index 0) is the number of courses taken in that department by majors (students whose major is that department), and the second integer (at index 1) is the number of courses taken in that department by non-majors (students whose major is undeclared or in a different department).

For example, using the `datalist` on the first page of this problem, `enrollmentsByKind(datalist)` returns the dictionary:

```
{'CompSci': [4, 7], 'Dance': [1, 1], 'Econ': [1, 5], 'Latin': [0,2],
'Math': [0, 5], 'Psy': [3, 1], 'Spanish': [0, 1]}
```

```
def enrollmentsByKind(datalist)
```

NETID: _____

Part E (9 pts) (9 minutes)

Write the function named **`majorConnections`** that has two parameters, one named **`datalist`**, which is a list of lists in the format described earlier, and another named **`dept`** that is a name of a department.

We repeat the format of parameter `datalist`, which is a list of lists. Each inner list has: (1) a string representing a student name, (2) an integer representing the student's id number (3) a string representing the home department of the student's major (possibly "Undeclared"), and (4) a list of departments of the courses taken by the student, in order from earliest to latest in the day, on Wednesdays.

This function returns a dictionary mapping each department to the number of times their courses are taken by students majoring in parameter department **`dept`**. Note that "Undeclared" is not a department and thus is not a key in the returned dictionary.

For example, using the `datalist` on the first page of this problem, `majorConnections(datalist, 'Psy')` returns the dictionary:

{`'Psy': 3, 'CompSci': 2, 'Econ': 1, 'Math': 2`}, since between the two Psy majors, they take 3 Psy courses, 2 CompSci courses, and so on.

```
def majorConnections(datalist, dept):
```

NETID: _____

BLANK PAGE IF YOU NEED MORE SPACE (MUST TURN IN)

Python Reference Sheet for Compsci 101, Exam 3, Fall 2025

DO NOT WRITE ANYTHING TO BE GRADED ON THESE REFERENCE SHEETS!!

Mathematical Operators		
Symbol	**Meaning**	**Example**
+	addition	4 + 5 = 9
-	subtraction	9 - 5 = 4
*	multiplication	3*5 = 15
/ and //	division	6/3 = 2.0 6/4 = 1.5 6//4 = 1
%	mod/remainder	5 % 3 = 2
**	exponentiation	3**2 = 9, 2**3 = 8
String Operators		
+	concatenation	"ab"+"cd"="abcd"
*	repeat	"xo"*3 = "xoxoxo"
Comparison Operators		
==	is equal to	3 == 3 is True
!=	is not equal to	3 != 3 is False
>=	is greater than or equal to	4 >= 3 is True
<=	is less than or equal to	4 <= 3 is False
>	is strictly greater than	4 > 3 is True
<	is strictly less than	3 < 3 is False
Boolean Operators		
`x=5`		
not	flips/negates the value of a bool	(not x == 5) is False
and	returns True only if both parts of it are True	(x > 3 and x < 7) is True (x > 3 and x > 7) is False
or	returns True if at least one part of it is True	(x < 3 or x > 7) is False (x < 3 or x < 7) is True
Type Conversion Functions		
int(x)	turn x into an integer value	int("123") == 123 int(5.8) == 5
	int can fail, e.g., int("abc") raises an error	
float(x)	turn x into an float value	float("2.46") == 2.46
	float can fail, e.g., float("abc") raises an error	
str(x)	turn x into a string value	str(432) == "432"
type(x)	the type of x	type(1) == int type(1.2) == float
String Index and Splicing		
`s="colorful""`		
		Example
s[x]	index a character	s[0] == 'c' s[-3] == 'f'

		s[5] == 'f'
s[x:y]	splice of string, substring from index x up to but not including index y	s[2:5] == 'lor' s[:5] == 'color' s[4:-1] == 'rfu' s[5:] == 'ful'

String Functions		
s="colorful"		
Name	**Returns**	**Example**
.find(str)	index of first occurrence	s.find("o") == 1
		s.find("e") == -1
.rfind(str)	index of last occurrence	s.rfind("o") == 3
		s.rfind("e") == -1
.index(str)	same as .find(str), error if str not in string	s.index("o") == 1
.count(str)	number of occurrences	s.count("o") == 2 s.count("r") == 1 s.count("e") == 0
.strip()	copy with leading/trailing whitespace removed	" big ".strip() == "big"
.split()	list of "words" in s	"big bad dog".split() == ["big","bad", "dog"]
.split(",")	list of "items " in s that are separated by a comma *In general can split on any string, not just a comma, e.g., s.split(":") will split on a colon and s.split("gat") will split on the string "gat".*	"this,old,man".split(",") == ["this", "old", "man"]
' '.join(lst)	concatenate elements of lst, a list of strings, separated by ' ' or any string	':'.join(['a','b','c']) == "a:b:c"
.startswith(str)	boolean if starts with string	s.startswith("color") == True s.startswith("cool") == False
.endswith(str)	boolean if ends with string	s.endswith("ful") == True s.endswith("color") == False
.upper()	uppercase of s	s.upper() == "COLORFUL"
.lower()	lowercase of s	"HELLO".lower() == "hello"
.isupper()	boolean is uppercase	'A'.isupper() == True 'a'.isupper() == False
.islower()	boolean is lowercase	'A'.islower() == False 'a'.islower() == True
.isalpha()	boolean is alphabetic character	'3'.isalpha() == False '?'.isalpha() == False 'z'.isalpha() == True
.capitalize()	capitalized s	s.capitalize() == "Colorful"
.replace(str1, str2)	replace all occurrences of str1 with str2	s.replace('o','y') == "cylyrful"
.replace(str1, str2,n)	replace the first n occurrences of str1 with str2	s.replace('o','y',1) == "cylorful"
Miscellaneous Functions		
help(x)	documentation for module x	
len(x)	length of sequence x, e.g., String or List	len("duke") == 4
list(str)	a list of the characters from string str	list("cards") == ['c','a','r','d','s']
sorted(x)	return list that is sorted version of sequence/iterable x, doesn't change x	sorted("cat") == ['a','c','t']

range(x)	a list of integers starting at 0 and going up to but not including x	range(5) == [0, 1, 2, 3, 4]
range(start, stop)	a list of integers starting at start and going up to but not including stop	range(3, 7) == [3, 4, 5, 6]
range(start, stop, inc)	a list of integers starting at start and going up to but not including stop with increment inc	range(3, 9, 2) == [3, 5, 7]
min(x, y, z)	minimum value of all arguments	min(3, 1, 2) == 1 min("z", "b", "a") == "a"
max(x, y, z)	maximum value of all arguments	max(3, 1, 2) == 3 max("z", "b", "a") == "z"
abs(x)	absolute value of the int or float x	abs(-33) == 33 abs(-33.5) == 33.5
List index, splicing and concatenation		
`lst =[3, 6, 8, 1, 7]`		
		Example
lst[x]	index an element	lst[0] == 3 lst[-1] == 7
lst[x:y]	splice of list, sublist from index x up to but not including index y	lst[1:3] == [6, 8] lst[:4] == [3, 6, 8, 1] lst[3:] == [1,7]
+ operator	concatenate two lists	[3,4] + [1,3,2] == [3,4,1,3,2]
List Functions		
`lst =[3, 6, 8, 1, 7]`		
sum(lst)	returns sum of elements in list `lst`	sum([1,2,4]) == 7
max(lst)	returns maximal element in `lst`	max([5,3,1,7,2]) == 7
lst.append(...)	append an element to lst, changing lst	[1,2,3].append(8) == [1,2,3,8]
lst.insert(pos,elt)	append `elt` to `lst` at position `pos`, changing `lst`	[1,2,3].insert(1,8) == [1,8,2,3]
lst.extend(lst2)	append every element of lst2 to lst	[1,2,3].extend([8,9]) == [1,2,3,8,9]
lst.remove(elt)	remove first occurence of `elt` from `lst`	[1,2,3,2,3,2].remove(2) == [1,3,2,3,2]
lst.sort()	sorts the elements of `lst`	lst = [3,6,8,1,7] lst.sort() lst is now [1, 3, 6, 7, 8]
lst.index(elt)	return index of elt in `lst`, error if elt not in lst	[1,5,3,8].index(5) == 1
lst.count(elt)	return number of occurrences of elt in `lst`	[1,2,1,2,3].count(1) == 2
lst.pop()	remove and return last element in `lst`, so has side-effect of altering list and returns value.	lst = [3,6,8,1,7] x = lst.pop() x is 7, lst is [3,6,8,1]
lst.pop(index)	remove and return element at position index in `lst`, so has side-effect of altering list and returns value. Default index is last value.	lst = [3,6,8,1,7] x = lst.pop(1) x is 6, lst is [3,8,1,7]
Math Functions (import math)		
math.pi	3.1415926535897931	
math.sqrt(num)	returns square root of num as float	math.sqrt(9) == 3.0
File Functions		
open("filename")	opens a file, returns file object	f = open("foo.txt")

open("filename",mode)	specify mode of 'r', 'a', 'w', return file object	f = open("foo.txt", "a")
f.read()	returns the entire file as one string	s = f.read()
Random Functions (import random)		
random.choice(list_of_choices)	returns a random element from list_of_choices. Gives an error if list_of_choices has length 0.	
random.randint(start, end)	Returns a random integer between start and end. Unlike range() and list slicing, the largest value it can return is end, not end-1.	
random.random()	Returns a random float between 0 and 1.	
Set Functions		
set(lst)	returns a set of the elements from list `lst`	
s.add(item)	adds the item into the set, and returns nothing.	
s.update(lst)	adds the elements in the list `lst` into the set, and returns nothing.	
s.remove(item)	removes the item from the set, error if item not there.	
s.union(t)	returns new set representing s UNION t, i.e., all elements in either s OR t, t can be any iterable (e.g., a list)	
s.intersection(t)	returns new set representing s INTERSECT t, i.e., only elements in both s AND t, t can be any iterable (e.g., a list)	
s.difference(t)	returns new set representing s difference t, i.e., elements in s that are not in t	
s.symmetric_difference(t)	returns new set representing elements in s or t, but not in both	
s | t	returns/evaluates to union of s and t, both must be sets.	
s & t	returns/evaluates to intersection of s and t, both must be sets.	
s - t	returns/evaluates to set with all elements in s that are *not* in t	
s ^ t	returns/evaluates to set with all elements from s and t that are *not* in both s and t	
Dictionary Functions		
d[key]	returns the value associated with `key`, error if `key` not in dictionary `d`	
d.get(key)	returns value associated with `key`, returns None if `key` not in dictionary `d`	
d.get(key,default)	returns value associated with `key`, returns `default` if `key` not in `d`	
d.keys()	returns a list/view of the keys in dictionary	
d.values()	returns a list/view of values in dictionary	
d.items()	returns a list/view of tuples, (key,item) pairs from dictionary	
d.update(dict)	updates the dictionary with another dictionary `dict`	
Lambda Functions		
`lst =[('c', [4, 2, 8]), ('h', [2,7,1,6]), ('b', [3, 9]) ]`		
f = lambda x : len(x[1]) y = sorted(lst, key=f)	y is `[ ('b', [3, 9]), ('c', [4, 2, 8]), ('h', [2,7,1,6]) ]` as it sorts tuples on the length of the lists in the index 1 position	
y = sorted(lst, key= lambda x: x[1])	y is `[ ('h', [2,7,1,6]), ('b', [3, 9]), ('c', [4, 2, 8]) ]` as it sorts tuples based on the first element in each list (the 2, 3, and 4)	
Image Library Functions		
Image.open(fname)	opens and returns image	
im.show()	displays image im	
im.getdata()	returns generator of all pixels in im	
im.putdata(pixlist)	modifies image by setting all pixels to pixlist	
im.size	returns tuple that is (width,height) of image	
Image.new("RGB",size)	creates and returns a new image with dimensions of tuple size	